Introduction

This module provides a helper DLL to the node package of this project (<MOD.021>). It enables access to many AMLEngine functionalities without using C# or the AMLEngine itself. In this project this module can be viewed as an interface between JavaScript(Node project) and the AMLEngine.dll, altough in theory it could be used from within different projects.

Requirements

This module implements parts of LF30 and LF40.

- LF30 states that the JavaScript wrapper should enable developers to use the functions of the AML.Engine.dll in their Node.js project.
- LF40 states that the project shall be able to be run in the JavaScript runtime environment Node.js.

Prerequisites

The user of this module needs :

- common necessities for coding (computer, elligible IDE, internet access, ...).
- a Windows operating system newer than Windows XP should be used. Different operating systems can work, but were not tested for coding and thus are not officially supported

Dependencies

1. This module needs an IDE to build dll files (e.g. Visual Studio Community)
2. Also you need to have the Aml.Engine.dll.

Communication with other modules/Interface

In this project, this module is used solely by the Node project (<MOD.0.21>). It gets imported into the project via Edge.js. The node project will be able to call a single function from this module as further described below.

Technical overview

File Structure

This module includes several files found in /src/Adapter/. Most of the functionality can be found in /src/Adapter/Adapter. Especially Class1.cs is important, because it handles the calls to it. These call

get handled in the Invoke function. The handling works through a switch statement, that simply matches a given function name to several options.
If there's the need to implement additional functions from the AMLEngine.dll this is the place to do so!
To compile the C# files into a dll, that can be used within a node project, you can simply build them via visual studio functionality. The created dll is in the /src/Adapter/Adapter/bin/Release (or */Debug depending on the build settings) folder.

Important Functions and Variables

As described above, the most important function is the Invoke function in the Class1.cs file. These are the current methods to invoke:

Function name	Description
"instancehierarchy_append"	This function appends a new internal element. It needs a payload with values for instance name, path to AML file to be modified and the content of the element.
"instancehierarchy_get"	This function returns a hierarchy element as string. It needs a path to the AML file and an indexer string, that contains either a number/string to find an internal element or a tuple (e.g. (Name:"Version", Value:"1.0")
"create_systemunitclass"	This function creates a new system unit class. Needed as payload are "unitclasslib_name", "unitclass_name" and "indexer" as above and the path to the aml file.
"create_interfaceclass"	This function creates a new interface class. It needs the same payload as the create_systemunitclass example above, but instead of "unitclasslib_name" it needs "interface_classname" and instead of "unitclass_name" it needs "iface_name"
"rename_element"	This function renames an internal element to signify its change in function for example. Payload consists of an "indexer" (e.g. name of an instance hierarchy element), an "ie" key containing the name of the element to change and a "newName" key containing the new name of the element.
"validate"	This function simply validates a file on a given path ("path" key must be given)
"repair"	This function validates and repairs a given document. The payload must contain a "path" variable to the aml file to repair

There are two parameters to give to this Invoke function:

Parameter	Prameter Description
Method name	This contains the name of the to be used function as defined in the switch statement as string.
Payload	This is a dynamic variable. It's a tuple containing several key-and-value pairs. Most notable keys are "path":"pathToAMLFile" and "indexer":a number/string to find an internal element or a tuple (e.g. (Name:"Version", Value:"1.0"))